https://www.joogn.de/sid.yannes.html Go MAY JUN 05 JAN
13 captures
5 Jun 2009 - 14 Sep 2019 2008 2009 2012
About this capture

Interview with Bob Yannes

This is an interview with the creator of the SID chip, namely **Bob Yannes**, who, after working for Commodore, co-founded the well-known synthesizer company **Ensoniq**. In the latter part of the interview he gives us a very detailed description of the SID's technology.

The interview was done via e-mail in August 1996 by Andreas Varga. Some questions came from King Fisher/Triad.

Did you foresee that people would actually treat your little VLSI-chip like an instrument?

Actually, I was an electronic music hobbyist before I started working for MOS Technology (one of Commodore's chip divisions at the time) and before I knew anything at all about VLSI chip design. One of the reasons I was hired was my knowledge of music synthesis was deemed valuable for future MOS/Commodore products. When I designed the SID chip, I was attempting to create a single-chip synthesizer voice which hopefully would find it's way into polyphonic/polytimbral synthesizers.

Are you aware of the existence of programs like SIDPLAY, PlaySID,... which emulate the SID chip up to the smallest click ?

I only recently became aware of them (through your website). I'm afraid I haven't thought much about SID in the last 15 years...I am constantly amazed and gratified at the number of people who have been positively affected by the SID chip and the Commodore 64 (which I also designed) and who continue to do productive things with them despite their "obsolescence".

Have you heard the tunes by Rob Hubbard, Martin Galway, Tim Follin, Jeroen Tel, and all the other composers ?

I'm afraid not, are recordings available in the US?

Did you believe this was possible to do with your chip?

Since I haven't heard them I'm not sure what we are talking about, however, I did design the SID chip with enough resolution to produce high-quality music. I was never able to refine the Signal-to-noise ratio to the level I wanted, though.

How much of the architecture in the SID inspired you when working with the Ensoniq synthesizers?

The SID chip was my first attempt at a phase-accumulating oscillator, which is the heart of all wavetable synthesis systems. Due to time constraints, the oscillators in SID were not multiplexed, therefore they took up a lot of chip area, constraining the number of voices I could fit on a chip. All ENSONIQ sound chips use a multiplexed oscillator which allows us to produce at least 32 voices per chip. Aside from that, little else of SID is to be found in our designs, which more closely resemble the Mountain Computer sound card for the Apple II (the basis of the Alpha Syntauri system). The DOC I chip (used in the Mirage and ESQ-1) was modeled on this sound card. Our current designs, which include waveform interpolation, digital filters and digital effects are new designs that aren't really based on anything other than our imaginations.

How big impact do you think the SID had on the synthesizer industry?

Well, I don't think it had much impact on the synthesizer industry. I remember once at Commodore that Sequential Circuits was interested in buying the chip, but nothing ever came of it. My intention in designing the chip (since MOS Technology was a merchant semiconductor house at the time and sold chips to the outside world) was to be able to sell the SID chip to synthesizer manufacturers. SID chip production was completely consumed by the Commodore 64 and by the time chips were readily available, I had left Commodore and never had the opportunity to improve the fidelity of the chip.

What would you have changed in the SIDs design, if you had a bigger budget from Commodore ?

The issue wasn't budget, it was development time and chip size constraints. The design/prototype/debug/production schedule of the SID chip, VIC II chip and Commodore 64 were incredibly tight (some would say impossibly tight)--we did things faster than Commodore had ever done before and were never able to repeat after! If I had had more time, I would have developed a proper MOS op-amp which would have eliminated the signal leakage which occurred when the volume of the voice was supposed to be zero. This lead to poor signal-to-noise ratio, although it could be dealt with by stopping the oscillator. It would also have greatly improved the filter, particularly in achieving high resonance. I originally planned to have an exponential look-up table to provide a direct translation for the equal-tempered scale, but it took up too much silicon and it was easy enough to do in software anyway.

The SID is very complex for its time. Why didn't you settle with an easier design ?

I thought the sound chips on the market (including those in the Atari computers) were primitive and obviously had been designed by people who knew nothing about music. As I said previously, I was attempting to create a synthesizer chip which could be used in professional synthesizers.

Do you still own a C64 (or another SID-equipped computer) ?

Sure, I have a couple of them (including the portable), but I honestly haven't turned them on in years.

Did Commodore ever plan to build an improved successor to the SID ?

I don't know. After I left I don't think there was anyone there who knew enough about music synthesis to do much more than improve the yield of the SID chip. I would have liked to have improved the SID chip before we had to release to production, but I doubt it would have made any difference to the success of the Commodore 64.

Can you give us a short overview of the SIDs internal architecture ?

It's pretty brute-force, I didn't have time to be elegant. Each "voice" consisted of an Oscillator, a Waveform Generator, a Waveform Selector, a Waveform D/A converter, a Multiplying D/A converter for amplitude control and an Envelope Generator for modulation. The analog output of each voice could be sent through a Multimode Analog Filter or bypass the filter and a final Multiplying D/A converter provided overall manual volume control.

As I recall, the Oscillator is a 24-bit phase-accumulating design of which thelower 16-bits are programmable for pitch control. The output of the accumulator goes directly to a D/A converter through a waveform selector. Normally, the output of a phase-accumulating oscillator would be used as an address into memory which contained a wavetable, but SID had to be entirely self-contained and there was no room at all for a wavetable on the chip.

The Sawtooth waveform was created by sending the upper 12-bits of the accumulator to the 12-bit Waveform D/A.

The Triangle waveform was created by using the MSB of the accumulator to invert the remaining upper 11 accumulator bits using EXOR gates. These 11 bits were then left-shifted (throwing away the MSB) and sent to the Waveform D/A (so the resolution of the triangle waveform was half that of the sawtooth, but the amplitude and frequency were the same).

The Pulse waveform was created by sending the upper 12-bits of the accumulator to a 12-bit digital comparator. The output of the comparator was either a one or a zero. This single output was then sent to all 12 bits of the Waveform D/A.

The Noise waveform was created using a 23-bit pseudo-random sequence generator (i.e., a shift register with specific outputs fed back to the input through combinatorial logic). The shift register was clocked by one of the intermediate bits of the accumulator to keep the frequency content of the noise waveform relatively the same as the pitched waveforms. The upper 12-bits of the shift register were sent to the Waveform D/A.

Since all of the waveforms were just digital bits, the Waveform Selector consisted of multiplexers that selected which waveform bits would be sent to the Waveform D/A. The multiplexers were single transistors and did not provide a "lock-out", allowing combinations of the waveforms to be selected. The combination was actually a logical ANDing of the bits of each waveform, which produced unpredictable results, so I didn't encourage this, especially since it could lock up the pseudo-random sequence generator by filling it with zeroes.

The output of the Waveform D/A (which was an analog voltage at this point) was fed into the reference input of an 8-bit multiplying D/A, creating a DCA (digitally-controlled-amplifier). The digital control word which modulated the amplitude of the waveform came from the Envelope Generator.

The Envelope Generator was simply an 8-bit up/down counter which, when triggered by the Gate bit, counted from 0 to 255 at the Attack rate, from 255 down to the programmed Sustain value at the Decay rate, remained at the Sustain value until the Gate bit was cleared then counted down from the Sustain value to 0 at the Release rate.

A programmable frequency divider was used to set the various rates (unfortunately I don't remember how many bits the divider was, either 12 or 16 bits). A small look-up table translated the 16 register-programmable values to the appropriate number to load into the frequency divider. Depending on what state the Envelope Generator was in (i.e. ADS or R), the appropriate register would be selected and that number would be translated and loaded into the divider. Obviously it would have been better to have individual bit control of the divider which would have provided great resolution for each rate, however I did not have enough silicon area for a lot of register bits. Using this approach, I was able to cram a wide range of rates into 4 bits, allowing the ADSR to be defined in two bytes instead of eight. The actual numbers in the look-up table were arrived at subjectively by setting up typical patches on a Sequential Circuits Pro-1 and measuring the envelope times by ear (which is why the available rates seem strange)!

In order to more closely model the exponential decay of sounds, another look-up table on the output of the Envelope Generator would sequentially divide the clock to the Envelope Generator by two at specific counts in the Decay and Release cycles. This created a piece-wise linear approximation of an exponential. I was particularly happy how well this worked considering the simplicity of the circuitry. The Attack, however, was linear, but this sounded fine.

A digital comparator was used for the Sustain function. The upper four bits of the Up/Down counter were compared to the programmed Sustain value and would stop the clock to the Envelope Generator when the counter counted down to the Sustain value. This created 16 linearly spaced sustain levels without havingto go through a look-up table translation between the 4-bit register value and the 8-bit Envelope Generator output. It also meant that sustain levels were adjustable in steps of 16. Again, more register bits would have provided higher resolution.

When the Gate bit was cleared, the clock would again be enabled, allowing the counter to count down to zero. Like an analog envelope generator, the SID Envelope Generator would track the Sustain level if it was changed to a lower value during the Sustain portion of the envelope, however, it would not count UP if the Sustain level were set higher.

The 8-bit output of the Envelope Generator was then sent to the Multiplying D/A converter to modulate the amplitude of the selected Oscillator Waveform (to be technically accurate, actually the waveform was modulating the output of the Envelope Generator, but the result is the same).

Hard Sync was accomplished by clearing the accumulator of an Oscillator based on the accumulator MSB of the previous oscillator.

Ring Modulation was accomplished by substituting the accumulator MSB of an oscillator in the EXOR function of the triangle waveform generator with the accumulator MSB of the previous oscillator. That is why the triangle waveform must be selected to use Ring Modulation.

The Filter was a classic multi-mode (state variable) VCF design. There was no way to create a variable transconductance amplifier in our NMOS process, so I simply used FETs as voltage-controlled resistors to control the cutoff frequency. An 11-bit D/A converter generates the control voltage for the FETs (it's actually a 12-bit D/A, but the LSB had no audible affect so I disconnected it!).

Filter resonance was controlled by a 4-bit weighted resistor ladder. Each bit would turn on one of the weighted resistors and allow a portion of the output to feed back to the input. The state-variable design provided simultaneous low-pass, band-pass and high-pass outputs. Analog switches selected which combination of outputs were sent to the final amplifier (a notch filter was created by enabling both the high and low-pass outputs simultaneously).

The filter is the worst part of SID because I could not create high-gain op-amps in NMOS, which were essential to a resonant filter. In addition, the resistance of the FETs varied considerably with processing, so different lots of SID chips had different cutoff frequency characteristics. I knew it wouldn't work very well, but it was better than nothing and I didn't have time to make it better.

Analog switches were also used to either route an Oscillator output through or around the filter to the final amplifier. The final amp was a 4-bit multiplying D/A converter which allowed the volume of the output signal to be controlled. By stopping an Oscillator, it was possible to apply a DC voltage to this D/A. Audio could then be created by having the microprocessor write the Final Volume register in real-time. Game programs often used this method to synthesize speech or play "sampled" sounds.

An external audio input could also be mixed in at the final amp or processed through the filter.

The Modulation registers were probably never used since they could easily be simulated in software without having to give up a voice. For novice programmers they provided a way to create vibrato or filter sweeps without having to write much code (just read the value from the modulation register and write it back to the frequency register). These registers just give microprocessor access to the upper 8 bits of the instantaneous value of the waveform and envelope of Voice 3. Since you probably wouldn't want to hear the modulation source in the audio output, an analog switch was provided to turn off the audio output of Voice 3.

Any other interesting tidbits or anecdotes ?

The funniest thing I remember was getting in a whole bunch of C-64 video games which had been written in Japan. The Japanese are so obsessed with technical specifications that they had written their code according to a SID spec. sheet (which I had written before SID prototypes even existed). Needless to say, the specs were not accurate. Rather than correct the obvious errors in their code, they produced games with out of tune sounds and filter settings that produced only quiet, muffled sound at the output. As far as they were concerned, it didn't matter that their code sounded all wrong, they had written their code correctly according to the spec. and that was all that mattered!

Last updated: Aug 30 1996